Logic and Inference: Rules

CSE 595 – Semantic Web
Stony Brook University
http://www.cs.stonybrook.edu/~cse595
Lecture Outline

• Monotonic Rules
• OWL2 RL: Description Logic Meets Rules
• Rule Interchange Format: RIF
• Semantic Web Rules Language (SWRL)
• Rules in SPARQL: SPIN
• Nonmonotonic Rules
  • Example: Brokered Trade
• Rule Markup Language (RuleML)
Introduction

• All we did until now are forms of knowledge representation (KR), like knowledge about the content of web resources, knowledge about the concepts of a domain of discourse and their relationships (ontology)

• Knowledge representation had been studied long before the emergence of the World Wide Web in the area of artificial intelligence and, before that, in philosophy
**Introduction**

- KR can be traced back to ancient Greece (because Aristotle is considered to be the father of logic)
- Logic is the foundation of knowledge representation, particularly in the form of *predicate logic* (also known as *first-order logic*)
Introduction

• Logic:
  • provides a high-level language in which knowledge can be expressed in a transparent way
  • has a high expressive power (maybe too high because it is intractable or undecidable in some cases)
  • has a well-understood formal semantics, which assigns an unambiguous meaning to logical statements
  • has a precise notion of **logical consequence**, which determines whether a statement follows semantically from a set of other statements (**premises**)
Introduction

- There exist proof systems that can automatically derive statements syntactically from a set of premises.
- There exist proof systems for which semantic logical consequence coincides with syntactic derivation within the proof system.
  - Proof systems should be sound (all derived statements follow semantically from the premises) and complete (all logical consequences of the premises can be derived in the proof system).
- Predicate logic is unique in the sense that sound and complete proof systems do exist - More expressive logics (higher-order logics) do not have such proof systems.
- It is possible to trace the proof that leads to a logical consequence, so logic can provide explanations for answers.
Introduction

• RDF and OWL2 profiles can be viewed as specializations of predicate logic:
  • One justification for the existence of such specialized languages is that they provide a syntax that fits well with the intended use (in our case, web languages based on tags).
  • Another justification is that they define reasonable subsets of logic where the computation is tractable (there is a trade-off between the expressive power and the computational complexity of certain logics: the more expressive the language, the less efficient the corresponding proof systems).
Introduction

- Most OWL variants correspond to a description logic, a subset of predicate logic for which efficient proof systems exist.
- Another subset of predicate logic with efficient proof systems comprises the Horn rule systems (also known as Horn logic or definite logic programs).
  - A rule has the form:
    \[ A_1, \ldots, A_n \rightarrow B \]
  - Where \( A_i \) and \( B \) are atomic formulas.
There are two intuitive ways of reading a Horn rule:

- **deductive rules:** If $A_1, \ldots, A_n$ are known to be true, then $B$ is also true
  - There are two ways of applying deductive rules:
    - from the body ($A_1, \ldots, A_n$) to the conclusion ($B$) (**forward chaining**)
    - from the conclusion (goal) to the body (**backward reasoning**)
- **reactive rules:** If the conditions $A_1, \ldots, A_n$ are true, then carry out the action $B$. 
Description logics and Horn logic are orthogonal in the sense that neither of them is a subset of the other.

For example, it is impossible to define the class of happy spouses as those who are married to their best friend in description logics, but this piece of knowledge can easily be represented using rules:

\[
\text{married}(X, Y), \text{bestFriend}(X, Y) \rightarrow \text{happySpouse}(X)
\]

On the other hand, rules cannot (in the general case) assert:
(a) negation/complement of classes
(b) disjunctive/union information (for instance, that a person is either a man or a woman)
(c) existential quantification (for instance, that all persons have a father).
Monotonic and nonmonotonic rules

• Predicate logic is **monotonic**: if a conclusion can be drawn, it remains valid even if new knowledge becomes available
  • Even if a rule uses negation,

R1 : If birthday, then special discount.
R2 : If **not birthday**, then not special discount.

it works properly in cases where the birthday is known
Imagine a customer who refuses to provide his birthday because of privacy concerns, then the preceding rules cannot be applied because their premises are not known.

R1 : If birthday, then special discount.
R2 : If not birthday, then not special discount.

R2' : If birthday is not known, then not special discount.

- R2' is not within the expressive power of predicate logic because its conclusion may become invalid if the customer’s birthday becomes known at a later stage and it happens to coincide with the purchase date.
- Adding knowledge later that invalidates some of the conclusions is called **nonmonotonic** because the addition of new information leads to a loss of a consequence.
Rules on the Semantic Web

- Rule technology has been around for decades, has found extensive use in practice, and has reached significant maturity
- led to a broad variety of approaches
- it is more difficult to standardize this area in the context of the (semantic) web
- A W3C working group has developed the Rule Interchange Format (RIF) standard
- Whereas RDF and OWL are languages meant for directly representing knowledge, RIF was designed primarily for the exchange of rules across different applications
  - For example, an online store might wish to make its pricing, refund, and privacy policies, which are expressed using rules, accessible to intelligent agents
Rules on the Semantic Web

- Due to the underlying aim of serving as an interchange format among different rule systems, RIF combines many of their features, and is quite complex.

- Those wishing to develop rule systems for the Semantic Web have various alternatives:
  - Rules over RDF can be expressed using SPARQL constructs. SPARQL is not a rule language, as basically it carries out one application of a rule.
  - SPIN is a rule system developed on top of SPARQL.
  - SWRL couples OWL DL functionalities with certain types of rules.
  - Model in terms of OWL but use rule technology for implementation purposes: OWL2 RL.
Example Monotonic Rules: Family

- Imagine a database of facts about some family relationships which contains facts about the following *base predicates*:
  
  \[
  \text{mother}(X, Y) \quad X \text{ is the mother of } Y
  \]
  
  \[
  \text{father}(X, Y) \quad X \text{ is the father of } Y
  \]
  
  \[
  \text{male}(X) \quad X \text{ is male}
  \]
  
  \[
  \text{female}(X) \quad X \text{ is female}
  \]
Example Monotonic Rules: Family

- We can infer further relationships using appropriate rules:
  - a parent is either a father or a mother.

\[
\text{mother}(X, Y) \rightarrow \text{parent}(X, Y).
\]
\[
\text{father}(X, Y) \rightarrow \text{parent}(X, Y).
\]

- a brother to be a male person sharing a parent:

\[
\text{male}(X), \ \text{parent}(P, X), \ \text{parent}(P, Y), \not\text{Same}(X, Y) \rightarrow \text{brother}(X, Y).
\]

- The predicate \(\not\text{Same}\) denotes inequality; we assume that such facts are kept in a database.

\[
\text{female}(X), \ \text{parent}(P, X), \ \text{parent}(P, Y), \not\text{Same}(X, Y) \rightarrow \text{sister}(X, Y).
\]
Example Monotonic Rules: Family

- An uncle is a brother of a parent:
  \[ \text{brother}(X, P), \text{parent}(P, Y) \rightarrow \text{uncle}(X, Y). \]

- A grandmother is the mother of a parent:
  \[ \text{mother}(X, P), \text{parent}(P, Y) \rightarrow \text{grandmother}(X, Y). \]

- An ancestor is either a parent or an ancestor of a parent:
  \[ \text{parent}(X, Y) \rightarrow \text{ancestor}(X, Y). \]
  \[ \text{ancestor}(X, P), \text{parent}(P, Y) \rightarrow \text{ancestor}(X, Y). \]
Monotonic Rules: Syntax

- Let us consider a simple rule stating that all loyal customers with ages over 60 are entitled to a special discount:

  \[ \text{loyalCustomer}(X), \text{age}(X) > 60 \rightarrow \text{discount}(X). \]

- Rules have:
  - variables, which are placeholders for values: \( X \)
  - constants, which denote fixed values: \( 60 \)
  - predicates, which relate objects: \( \text{loyalCustomer}, > \)
  - function symbols, which denote a value, when applied to certain arguments: \( \text{age} \)

- In case no function symbols are used, we discuss function-free (Horn) logic.
Rules

• A rule has the form:

    \[ B_1, \ldots, B_n \rightarrow A \]

where \( A \) and \( B_i \) are atomic formulas

• \( A \) is the **head** of the rule

• \( B_i \) are the **premises** of the rule

• The set \( \{B_1, \ldots, B_n\} \) is referred to as the **body** of the rule

• The commas in the rule body are read conjunctively:

    if \( B_1 \) and \( B_2 \) and \( \ldots \) and \( B_n \) are true, then \( A \) is also true

    • (or equivalently, to prove \( A \) it is sufficient to prove all of \( B_1, \ldots, B_n \))
Rules

- Note that variables may occur in $A, B_1, \ldots, B_n$.
  - For example,

    $$\text{loyalCustomer}(X), \text{age}(X) > 60 \rightarrow \text{discount}(X).$$

    is applied for any customer: if a customer happens to be loyal and over 60, then they gets the discount.

- The variable $X$ is implicitly universally quantified (using $\forall X$)

- In general, all variables occurring in a rule are implicitly universally quantified.
A rule $r$:

$$B_1, \ldots, B_n \rightarrow A$$

is interpreted as the following formula, denoted by $pl(r)$:

$$\forall X_1 \ldots \forall X_k \ ( (B_1 \land \ldots \land B_n) \rightarrow A)$$

or equivalently,

$$\forall X_1 \ldots \forall X_k \ (A \lor \neg B_1 \lor \ldots \lor \neg B_n)$$

where $X_1, \ldots, X_k$ are all variables occurring in $A, B_1, \ldots, B_n$. 
Logic Programs

• A fact is an atomic formula, such as `loyalCustomer(a345678)`. which says that the customer with ID `a345678` is loyal.
  • If there are variables in a fact, then they are implicitly universally quantified.

• A logic program `P` is a finite set of facts and rules.
  • Its predicate logic translation `pl(P)` is the set of all predicate logic interpretations of rules and facts in `P`. 
Logic Programs

- A **goal** or **query** $G$ asked to a logic program has the form $B_1, \ldots, B_n \rightarrow$
- If $n = 0$, we have the empty goal $\Box$.

- The interpretation of a goal is:
  \[ \forall X_1 \ldots \forall X_k (\neg B_1 \lor \ldots \lor \neg B_n) \]
  where $X_1, \ldots, X_k$ are all variables occurring in $B_1, \ldots, B_n$
Logic Programs

- The formula is equivalent to
  \[
  \forall X_1 \ldots \forall X_k (\text{false} \lor \neg B_1 \lor \ldots \lor \neg B_n)
  \]
  so the missing rule head can be thought of as a contradiction \text{false}.

- An equivalent representation in predicate logic is:
  \[
  \neg \exists X_1 \ldots \exists X_k (B_1 \land \ldots \land B_n)
  \]
Logic Programs

• Suppose we know the fact

  \( p(a) \).

and we have the goal

\( p(X) \rightarrow \)

• we want to know whether there is a value for which \( p \) is true
• We expect a positive answer because of the fact \( p(a) \).
• Thus \( p(X) \) is existentially quantified
• Why do we negate the formula?
  • The explanation is that we use a proof technique from mathematics called \textit{proof by contradiction}
  • This technique proves that a statement A follows from a statement B by assuming that A is false and deriving a contradiction when combined with B. Then A must follow from B.
Logic Programs

• In logic programming we prove that a goal can be answered positively by negating the goal and proving that we get a contradiction using the logic program.
  • For example, given the logic program
    \[ p(a). \]
    and the goal
    \[ \neg \exists x p(x) \]
    we get a logical contradiction: the second formula says that no element has the property \( p \), but the first formula says that the value of \( a \) does have the property \( p \).
    Thus \( \neg \exists x p(x) \) follows from \( p(a) \).
Monotonic Rules: Semantics

- Given a logic program $P$ and a query $B_1, \ldots, B_n \rightarrow$ with the variables $x_1, \ldots, x_k$, we answer positively if, and only if,
  $$pl(P) \models \exists x_1 \ldots \exists x_k (B_1 \land \ldots \land B_n)$$
  or equivalently, if
  $$pl(P) \cup \{\neg \exists x_1 \ldots \exists x_k (B_1 \land \ldots \land B_n)\}$$
  is unsatisfiable

- We give a positive answer if the predicate logic representation of the program $P$, together with the predicate logic interpretation of the query, is unsatisfiable (a contradiction).
Monotonic Rules: Semantics

• Predicate Logic Semantics
  • A predicate logic model, $A$, consists of
    • a domain $\text{dom}(A)$, a nonempty set of objects about which the formulas make statements
    • an element from the domain for each constant
    • a concrete function on $\text{dom}(A)$ for every function symbol
    • a concrete relation on $\text{dom}(A)$ for every predicate
  • When the symbol $=$ is used to denote equality (i.e., its interpretation is fixed), we talk of Horn logic with equality
  • Logical connectives $\neg$, $\lor$, $\land$, $\rightarrow$, $\forall$, $\exists$
  • A formula $\phi$ follows from a set $M$ of formulas if $\phi$ is true in all models $A$ in which $M$ is true (that is, all formulas in $M$ are true in $A$).
Monotonic Rules: Semantics

• A formula $\phi$ follows from a set $M$ of formulas if $\phi$ is true in all models $A$ in which $M$ is true (that is, all formulas in $M$ are true in $A$).

• Regardless of how we interpret the constants, predicates, and function symbols occurring in $P$ and the query, once the predicate logic interpretation of $P$ is true, $\exists x_1 \ldots \exists x_k (B_1 \land \ldots \land B_n)$ must be true: that is, there are values for the variables $x_1, \ldots, x_k$ such that all atomic formulas $B_i$ become true.
Monotonic Rules: Semantics

- Suppose \( P \) is the program
  \[
  p(a) . \\
  p(X) \rightarrow q(X) .
  \]
- Consider the query
  \[
  q(X) \rightarrow
  \]
  - \( q(a) \) follows from \( pl(P) \)
  - \( \exists X q(X) \) follows from \( pl(P) \)
  - \( pl(P) \cup \{ \neg \exists X q(X) \} \) is unsatisfiable
- If we consider the query
  \[
  q(b) \rightarrow
  \]
  then we give a negative answer because \( q(b) \) does not follow from \( pl(P) \)
Least Herbrand Model Semantics

• Instead of considering any domain \( \text{dom}(A) \), we can consider only the names in the program (predicate names, constants, functors).

• Then we have **Herbrand semantics**:  
  • Given an alphabet \( A \), the set of all ground terms constructed from the constant and function symbols of \( A \) is called the **Herbrand Universe** of \( A \) (denoted by \( U_A \)).

• Consider the program:

\[
\begin{align*}
p(zero) & . \\
p(s(s(X))) & \leftarrow p(X) .
\end{align*}
\]

• The Herbrand Universe of the program's alphabet is:

\[ U_A = \{ \text{zero}, s(\text{zero}), s(s(\text{zero})), \ldots \} \]
Consider the "relations" program:

\[
\text{parent}(\text{pam}, \text{bob}). \quad \text{parent}(\text{bob}, \text{ann}).
\text{parent}(\text{tom}, \text{bob}). \quad \text{parent}(\text{bob}, \text{pat}).
\text{parent}(\text{tom}, \text{liz}). \quad \text{parent}(\text{pat}, \text{Jim}).
\\text{grandparent}(X,Y) : - \\
\quad \text{parent}(X,Z), \text{parent}(Z,Y).
\]

The Herbrand Universe of the program's alphabet is:

\[
U_A = \{\text{pam, bob, tom, liz, ann, pat, jim}\}
\]
Least Herbrand Model Semantics

- Given an alphabet $A$, the set of all ground atomic formulas over $A$ is called the Herbrand Base of $A$ (denoted by $B_A$).
- Consider the program:
  
  $p(\text{zero}).$
  
  $p(s(s(X))) \leftarrow p(X).$

- The Herbrand Base of the program's alphabet is:
  
  $B_A = \{ p(\text{zero}), p(s(\text{zero})), p(s(s(\text{zero}))), ... \}$
Least Herbrand Model Semantics

• Consider the "relations" program:

\[
\begin{align*}
\text{parent}(pam, bob) &. & \text{parent}(bob, ann) &. \\
\text{parent}(tom, bob) &. & \text{parent}(bob, pat) &. \\
\text{parent}(tom, liz) &. & \text{parent}(pat, jim) &. \\
\text{grandparent}(X,Y) :&= & \text{parent}(X,Z), & \text{parent}(Z,Y).
\end{align*}
\]

• The Herbrand Base of the program's alphabet is:

\[B_A = \{ \text{parent}(pam, pam), \text{parent}(pam, bob), \text{parent}(pam, tom), \ldots, \text{parent}(bob, pam), \ldots, \text{grandparent}(pam,pam), \ldots, \text{grandparent}(bob,pam), \ldots \}\]
Least Herbrand Model Semantics

- A **Herbrand Interpretation** of a program $P$ is an interpretation $I$ such that:
  - The domain of the interpretation: $|I| = U_P$
  - For every constant $c$: $c_I = c$
  - For every function symbol $f/n$: $f(x_1,\ldots,x_n)=f(x_1,\ldots,x_n)$
  - For every predicate symbol $p/n$: $p_I \subseteq (U_P)^n$ (i.e. some subset of $n$-tuples of ground terms)
- A **Herbrand Model** of a program $P$ is a Herbrand interpretation that is a model of $P$. 
Least Herbrand Model Semantics

• All Herbrand interpretations of a program give the same "meaning" to the constant and function symbols.
• Different Herbrand interpretations differ only in the "meaning" they give to the predicate symbols.
• We often write a Herbrand model simply by listing the subset of the Herbrand base that is true in the model.

Example: Consider our numbers program, where
\{p(\texttt{zero}), p(s(s(\texttt{zero})))) , p(s(s(s(s(\texttt{zero}))))), \ldots \} represents the Herbrand model that treats
\[p_I=\{\texttt{zero}, s(s(\texttt{zero})), s(s(s(s(\texttt{zero})))) , \ldots \} \]
as the meaning of \(p\).
Sufficiency of Herbrand Models

- Let P be a definite program. If I' is a model of P then
  \[ I = \{ A \in Bp \mid I' \models A \} \]
  is a Herbrand model of P.

Proof (by contradiction):
Let I be a Herbrand interpretation.
Assume that I' is a model of P but I is not a model.
Then there is some ground instance of a clause in P:
\[ A_0 :\leftarrow A_1, \ldots, A_n. \]
which is not true in I i.e., \( I \models A_1, \ldots, I \models A_n \) but \( I \not\models A_0 \)
By definition of I then, \( I' \models A_1, \ldots, I' \models A_n \) but \( I' \not\models A_0 \)
Thus, I' is not a model of P, which contradicts our earlier assumption.
Definite programs only

- Let P be a definite program. If I' is a model of P then
  \[ I = \{ A \in Bp \mid I' \models A \} \]
  is a Herbrand model of P.

- This property holds only for definite programs!
  
  - Consider \( P = \{ \neg p(a), \exists X. p(X) \} \)
  
    - There are two Herbrand interpretations: \( I_1 = \{ p(a) \} \) and
      \( I_2 = \{ \} \)
      
      - The first is not a model of P since \( I_1 \not\models \neg p(a) \).
      - The second is not a model of P since \( I_2 \not\models \exists X. p(X) \)

  - But there is a non-Herbrand model I:
    
    - \( | I | = \mathbb{N} \), the set of natural numbers
    - \( a_I = 0 \)
    - \( p_I = \text{“is odd”} \)
Properties of Herbrand Models

1) If $M$ is a set of Herbrand Models of a definite program $P$, then $\bigcap M$ is also a Herbrand Model of $P$.

2) For every definite program $P$ there is a unique least model $M_P$ such that:
   - $M_P$ is a Herbrand Model of $P$ and,
   - for every Herbrand Model $M$, $M_P \subseteq M$.

3) For any definite program, if every Herbrand Model of $P$ is also a Herbrand Model of $F$, then $P \models F$.

4) $M_P = \text{the set of all ground logical consequences of } P$. 
Properties of Herbrand Models

• If $M_1$ and $M_2$ are Herbrand models of $P$, then $M = M_1 \cap M_2$ is a model of $P$.

• Assume $M$ is not a model.

• Then there is some clause $A_0 : \neg A_1, \ldots, A_n$ such that $M \models A_1, \ldots, M \models A_n$ but $M \not\models A_0$.

• Which means $A_0 \notin M_1$ or $A_0 \notin M_2$.

• But $A_1, \ldots, A_n \in M_1$ as well as $M_2$.

• Hence one of $M_1$ or $M_2$ is not a model.
Properties of Herbrand Models

- There is a unique least Herbrand model
- Let $M_1$ and $M_2$ are two incomparable minimal Herbrand models, i.e., $M = M_1 \cap M_2$ is also a Herbrand model (previous theorem), and $M \subseteq M_1$ and $M \subseteq M_2$
- Thus $M_1$ and $M_2$ are not minimal.
Least Herbrand Model

• The **least Herbrand model** $M_p$ of a definite program $P$ is the set of all ground logical consequences of the program.

$$M_p = \{ A \in B_p \mid P \models A \}$$

• First, $M_p \supseteq \{ A \in B_p \mid P \models A \}$:
  • By definition of logical consequence, $P \models A$ means that $A$ has to be in every model of $P$ and hence also in the least Herbrand model.
Least Herbrand Model

- Second, $\mathcal{M}_p \subseteq \{A \in B_p \mid P \models A\}$:
  - If $\mathcal{M}_p \not\models A$ then $A$ is in every Herbrand model of $P$.
  - But assume there is some model $I' \not\models \neg A$.
  - By sufficiency of Herbrand models, there is some Herbrand model $I$ such that $I \models \neg A$.
  - Hence $A$ is not in some Herbrand model, and hence is not in $\mathcal{M}_p$. 
Finding the Least Herbrand Model

• **Immediate consequence operator:**
  - Given $I \subseteq B_p$, construct $I'$ such that
    $$I' = \{ A_0 \in B_p \mid A_0 \leftarrow A_1, \ldots, A_n \text{ is a ground instance of a clause in } P \text{ and } A_1, \ldots, A_n \in I \}$$
  - $I'$ is said to be the *immediate consequence of* $I$.
  - Written as $I' = T_p(I)$, $T_p$ is called the *immediate consequence operator*.
  - Consider the sequence:
    $$\emptyset, T_p(\emptyset), T_p(T_p(\emptyset)), \ldots, T_p^i(\emptyset), \ldots$$
  - $M_p \supseteq T_p^i(\emptyset)$ for all $i$.
  - Let $T_p \uparrow \omega = \bigcup_{i=0}^{\infty} T_p^i(\emptyset)$
  - Then $M_p \subseteq T_p \uparrow \omega$
Computing Least Herbrand Models: An Example

\[
\begin{align*}
\text{parent}(pam, bob). \\
\text{parent}(tom, bob). \\
\text{parent}(tom, liz). \\
\text{parent}(bob, ann). \\
\text{parent}(bob, pat). \\
\text{parent}(pat, jim). \\
\text{anc}(X, Y) :- \\
\hspace{1em} \text{parent}(X, Y). \\
\text{anc}(X, Y) :- \\
\hspace{1em} \text{parent}(X, Z), \\
\hspace{2em} \text{anc}(Z, Y). \\
\end{align*}
\]

\[
\begin{array}{|c|c|}
\hline
M_1 & \emptyset \\
\hline
M_2 = T_P(M_1) & \{ \text{parent}(pam, bob), \\
& \hspace{1em} \text{parent}(tom, bob), \\
& \hspace{1em} \text{parent}(tom, liz), \\
& \hspace{1em} \text{parent}(bob, ann), \\
& \hspace{1em} \text{parent}(bob, pat), \\
& \hspace{1em} \text{parent}(pat, jim) \} \\
\hline
M_3 = T_P(M_2) & \{ \text{anc}(pam, bob), \hspace{1em} \text{anc}(tom, bob), \\
& \hspace{1em} \text{anc}(tom, liz), \hspace{1em} \text{anc}(bob, ann), \\
& \hspace{1em} \text{anc}(bob, pat), \hspace{1em} \text{anc}(pat, jim) \} \cup M_2 \\
\hline
M_4 = T_P(M_3) & \{ \text{anc}(pam, ann), \hspace{1em} \text{anc}(pam, pat), \\
& \hspace{1em} \text{anc}(tom, ann), \hspace{1em} \text{anc}(tom, pat), \\
& \hspace{1em} \text{anc}(bob, jim) \} \cup M_3 \\
\hline
M_5 = T_P(M_4) & \{ \text{anc}(pam, jim), \hspace{1em} \{ \text{anc}(tom, jim) \} \} \cup M_4 \\
\hline
M_6 = T_P(M_5) & M_5 \\
\hline
\end{array}
\]
Suppose we know the fact
\[ p(a) \]
and we have the goal
\[ p(X) \rightarrow \]

- Responding **true** to parametrized queries is correct, but not satisfactory
- The appropriate answer is a substitution \( \{X/a\} \) which gives an instantiation for \( X \), making the answer positive.
- The constant \( a \) is called a *ground witness*
- Given two facts: \( p(a) \) and \( p(b) \). there are two ground witnesses to the same query: \( a \) and \( b \).
Ground and Parameterized Witnesses

- Ground witnesses are not always the optimal answer
  - Consider the logic program:
    \[
    \text{add}(X, 0, X)
    \]
    \[
    \text{add}(X, Y, Z) \rightarrow \text{add}(X, s(Y), s(Z))
    \]
  - Responding true to parametrized queries is correct, but not satisfactory
    - The appropriate answer is a substitution \(\{X/a\}\) which gives an instantiation for \(X\), making the answer positive.
    - The constant \(a\) is called a ground witness
  - Given two facts: \(p(a)\). and \(p(b)\). there are two ground witnesses to the same query: \(a\) and \(b\).